39장 DOM

> **DOM**(Document Object Model)은 HTML 문서의 계층적 구조와 정보를 표현하며
> 이를 제어할 수 있는 API
>
> **⇒ 프로퍼티와 메서드를 제공하는 트리 자료구조**

39.1 노드

- HTML 요소는 HTML 문서를 구성하는 개별적인 요소
- HTML 요소는 렌더링 엔진에 의해 파싱되어 DOM을 구성하는 요소 노드 객체로 변환
- HTML 요소 간에는 중첩 관계를 가질 수 있으며 이러한 관계를 반영하여 HTML 요소를 객체화한 모든 노드 객체들을 트리 자료 구조로 구성

39.1.2 노드 객체의 타입

- 노드 객체는 상속 구조를 지님

1. 문서 노드(`document node`)
    - DOM 트리의 최상위에 존재하는 루트 노드
        - document 객체를 가리킴
    - document 객체는 브라우저가 렌더링한 HTML문서 전체를 가리키는 객체
    - window.document 혹은 document로 참조 가능
    - script 태그로 자바스크립트 파일이 분리되어 있어도 전역 객체 window.document에 바인딩된 객체를 따름 (HTML 문서당 document객체는 유일)
    - document객체는 DOM트리의 루트 노드 이므로 DOM 트리의 노드들에 접근하기 위한 진입점 역할
2. 요소 노드(`element node`)

- HTML 요소를 가리키는 객체
- 각 HTML 요소들은 상속관계를 가지므로, 요소 노드는 문소의 구조를 표현

3. 어트리뷰트 노드(`attribute node`)
   - HTML 요소의 어트리뷰트를 가리키는 객체
   - 지정된 HTML 요소의 노드와 연결
   - 요소 노드에만 연결되어 있고 부모 노드를 가지지 않음
   - 따라서, 어트리뷰트 노드에 접근하려면 요소 노드에 먼저 접근 해야함
4. 텍스트 노드(`text node`)
   - HTML 요소의 텍스트를 가리키는 객체
   - 요소 노드의 자식 노드
   - 자식 노드를 가질 수 없는 리프 노드(leaf node)
     - 즉, DOM 트리의 최종단
   - 따라서 , 텍스트 노드에 접근하려면 부모 노드인 요소 노드에 접근해야 함

39.1.3 노드 객체의 상속 구조

- DOM 을 구성하는 노드 객체는 표준 빌트인 객체가 아니라 브라우저 환경에서 추가적으로 제공하는 **호스트 객체**
- 노드 객체도 자바스크립트 객체이므로 프로토타입에 의한 상속 구조를 가짐
- 모든 노드 객체는 `Object`, `EventTarget`, `Node` 인터페이스를 상속
- HTML 요소의 종류에 따라 고유의 기능을 가지기도 한다
- `DOM`은 `HTML` 문서의 계층적 구조와 정보를 표현하는 것은 물론 노드 객체의 종류
  - → 노드 타입에 따라 필요한 기능을 프로퍼티와 메서드의 집합인 `DOM API`로 제공
  - → 이 `DOM API`를 통해 `HTML`의 구조나 내용 또는 스타일 등을 동적으로 조작 가능

39.2 요소 취득 노드

39.2.1 id를 이용한 요소 노드 취득

- `Document.prototype.getElementById` 메서드 사용

- 인수로 전달된 id 값을 갖는 첫 번째 요소 노드만 반환

39.2.2 태그 이름을 이용한 요소 노드 취득

- `getElementsByTagName` 메서드 사용
  - `Document.prototype.getElementsByTagName` : 루트 노드인 문서노드(`document`)를 통해 호출
  - `Element.prototype.getElementsByTagName` : 특정 요소 노드를 통해 호출 (특정 요소 노드의 자식 중에서 탐색)
- `DOM` 컬렉션 객체인 `HTMLCollection` 객체 반환(유사 배열 이면서 이터러블)

39.2.3 class를 이용한 요소 노드 취득

- `getElementsByClassName` 메서드 사용
- 인수로 전달한 class 값을 갖는 모든 요소 노드 반환
- 공백으로 여러 class 지정 가능
- `DOM` 컬렉션 객체인 `HTMLCollection` 객체 반환(유사 배열 이면서 이터러블)

39.2.4 CSS 선택자를 이용한 요소 노드 취득

```
/* 전체 선택자: 모든 요소를 선택 */
* { ... }
/* 태그 선택자: 모든 p 태그 요소를 모두 선택 */
p { ... }
/* id 선택자: id 값이 'foo'인 요소를 모두 선택 */
#foo { ... }
/* class 선택자: class 값이 'foo'인 요소를 모두 선택 */
.foo { ... }
/* 어트리뷰트 선택자: input 요소 중에 type 어트리뷰트 값이 'text'인 요소를 모두 선택 */
input[type=text] { ... }
/* 후손 선택자: div 요소의 후손 요소 중 p 요소를 모두 선택 */
div p { ... }
/* 자식 선택자: div 요소의 자식 요소 중 p 요소를 모두 선택 */
div > p { ... }
```

```
/* 인접 형제 선택자: p 요소의 형제 요소 중에 p 요소 바로 뒤에 위치하는 ul 요소를 선택 */
p + ul { ... }
/* 일반 형제 선택자: p 요소의 형제 요소 중에 p 요소 뒤에 위치하는 ul 요소를 모두 선택 */
p ~ ul { ... }
/* 가상 클래스 선택자: hover 상태인 a 요소를 모두 선택 */
a:hover { ... }
/* 가상 요소 선택자: p 요소의 콘텐츠의 앞에 위치하는 공간을 선택
   일반적으로 content 프로퍼티와 함께 사용된다. */
p::before { ... }
```

💡querySelector()

- 인수로 전달할 CSS 선택자를 만족시키는 요소 노드가 여러 개인 경우 첫 번째 요소 노드만 반환
- 인수로 전달된 CSS 선택자를 만족시키는 요소 노드가 존재하지 않는 경우 `null`을 반환
- 인수로 전달한 CSS 선택자가 문법에 맞지 않는 경우 `DOMException` 에러 발생
- `Document.prototype` / `Element.prototype` 에 정의 된 메서드로 나뉨

💡querySelectorAll()

- 인수로 전달한 CSS 선택자를 만족 시키는 모든 요소 노드를 탐색하여 반환
- 여러개의 요소 노드 객체를 갖는 `DOM` 컬렉션 객체인 `NodeList` 를 반환
- `NodeList`는 유사 배열 객체 이면서 이터러블
- 요소가 존재하지 않는 경우 빈 `NodeList` 객체 반환
- 문법에 맞지 않는 경우 `DOMException`

39.2.5 특정 요소 취득할 수 있는지 확인

- `Element.prototype.matches`메서드 사용
- 인수로 전달한 CSS 선택자를 통해 특정 요소 노드를 취득할 수 있는지 확인
- 이벤트 위임을 사용할 때 유용

```
const $apple = document.querySelector('.apple');
```

```
// $apple 노드는 '#fruits > li.apple'로 취득할 수 있다.
console.log($apple.matches('#fruits > li.apple'));  // true

// $apple 노드는 '#fruits > li.banana'로 취득할 수 없다.
console.log($apple.matches('#fruits > li.banana')); // fals
e
```

39.2.6 HTMLCollection 과 NodeList

💡HTMLCollection

- `HTMLCollection`은 객체의 상태 변화를 실시간으로 반영
- 살아 있는 객체라고 부르기도 함
- 상태가 즉시 반영 되기 때문에 에러가 나기도 함
  - 이를 해결하기 위해 배열로 변환하여 사용 권장

```
// class 값이 'red'인 요소 객체를 모두 HTMLCollection 으로 반환
const $elems = document.getElementsByClassName('red');// 객
체 상태가 실시간으로 반영되면서 for문 으로 사용할 경우 원하는 대로 조작
하기 어려움
// 배열로 변환해서 사용

[...$elems].forEach(ele => ele.className = 'blue');
```

💡NodeList

- 실시간으로 객체의 상태 변경을 반영하지 않는 객체
- `NodeList.prototype.forEach` 메서드를 상속 받아 사용할 수 있음
- 예외로, `childNodes` 프로퍼티가 반환하는 `NodeList` 객체는 `HTMLCollection`과 같이 상태가 실시간으로 반영
- 노드 객체의 상태 변경과 상관없이 사용하려면 HTMLColleciton 이나 NodeList 객체를

  배열로 변환해서 사용을 권장
- 두 객체는 모두 유사 배열이면서 이터러블
  - → 스프레드 문법으로 배열로 변환 가능

39.3 노드 탐색

- DOM 트리 상의 노드를 탐색 할 수 있도록 Node, Element 인터스페이스는 트리 탐색 프로퍼티를 제공
- `Node.prototype` 이 제공하는 프로퍼티
  - `parentNode` : 부모 노드 탐색 (부모 노드는 텍스트 노드가 될 수 없음)
  - `previousSibling` : 이전 형제 노드 탐색하여 반환, 요소 노드 또는 텍스트 노드를 반환
  - `firstChild` : 첫 번째 자식 노드 반환 , 요소 노드 또 는 텍스트 노드
  - `childNodes` : 자식 노드를 모두 탐색 하여 `NodeList`로 반환, 텍스트 노드 포함 될 수 있음
- `Element.prototype` 이 제공하는 프로퍼티
  - `previousElementSibling` : 이전 형제 노드 반환, 요소 노드만 반환
  - `nextElementSibling` : 자신의 다음 형제 요소 노드 반환, 요소 노드만 반환
  - `children` : 자식 노드 중에서 요소 노드만 모두 탐색하여 `HTMLCollection` 으로 반환 (텍스트 노드 포함 X)

39.4 노드 정보 취득

💡Node.prototype.nodeType

- 노드 객체의 종류
  - 노드 타입을 나타내는 상수를 반환
- 노드 타입 상수는 `Node`에 정의되어 있음
- `Node.ELEMENT_NODE` : 요소 노드 타입을 나타내는 상수 `1`을 반환
- `Node.TEXT_NODE` : 텍스트 노드 타입을 나타내는 상수 `3`을 반환
- `Node.DOCUMENT_NODE` : 문서 노드 타입을 나타내는 상수 `9`를 반환

💡Node.prototype.nodeName

- 노드의 이름을 문자열로 반환
- 요소 노드: 대문자 문자열로 태그 이름(`"UL"`, `"LI"` 등)을 반환
- 텍스트 노트: 문자열 `"#text"`를 반환
- 문서 노드: 문자열 `"#document"`를 반환

39.5 요소 노드의 텍스트 조작

39.5.1 nodeValue

- 노드 객체의 값(텍스트 노드의 텍스트)을 반환
- 문서 노드나 요소 노드는 `nodeValue` 참조하면 `null` 반환

```
<!DOCTYPE html>
<html>
    <body>
        <div id="foo">Hello</div>
    </body>
    <script>
        // 1. #foo 요소 노드의 자식 노드인 텍스트 노드를 취득한다.
    const $textNode = document.getElementById('foo').firstC
hild;

        // 2. nodeValue 프로퍼티를 사용하여 텍스트 노드의 값을 변경
한다.
    $textNode.nodeValue = 'World';

    console.log($textNode.nodeValue); // World
    </script>
</html>
```

39.5.2 textContent

- 요소 노드의 텍스트와 자손 노드의 텍스트를 모두 취득
    - textContent는 요소노드의 콘텐츠 영역(시작 태그와 종료태그 사이)의 모든 텍스트를 반환

- 이때 HTML 마크업은 무시

```
<!DOCTYPE html>
<html>
    <body>
        <div id="foo">Hello <span>world!</span></div>
    </body>
    <script>
    // #foo 요소 노드의 텍스트를 모두 취득한다. 이때 HTML 마크업은 무시된다.
    console.log(document.getElementById('foo').textContent); // Hello world!
  </script>
</html>
```

39.6 DOM 조작

- 새로운 노드를 DOM에 추가하거나, 기존 노드를 삭제, 교체 등의 조작 가능

39.6.1 innerHTML

- 요소 노드의 콘텐츠 영역 사이에 포함된 모든 HTML 마크업을 문자열로 반환
- innerHTML 프로퍼티에 문자열을 할당하면 요소노드의 모든 자식 노드가 제거되고 할당한 문자열이 파싱되어 DOM에 반영

```
<!DOCTYPE html>
<html>
    <body>
        <ul id="fruits">
            <li class="apple">Apple</li>
        </ul>
    </body>
    <script>
    const $fruits = document.getElementById('fruits');

    // 노드 추가
```

```
    $fruits.innerHTML += '<li class="banana">Banana</li>';

    // 노드 교체
    $fruits.innerHTML = '<li class="orange">Orange</li>';

    // 노드 삭제
    $fruits.innerHTML = '';
  </script>
</html>
```

- innerHTML 단점
  - 크로스 사이트 스크립트 공격에 취약
  - 모든 자식요소를 제거하고 할당하기 때문에 비효율적
  - 새로운 요소를 삽입할 때 위치를 지정할 수 없음

39.6.2 insertAdjacentHTML

- 기존 요소를 제거하지 않으면서 위치를 지정해 새로운 요소를 삽입
- 첫 번재 인수로 위치를 지정 (`beforebegin`, `afterbegin`, `beforend`, `afterend`)

```
<!DOCTYPE html>
<html>
    <body>
        <!-- beforebegin -->
    <div id="foo">
        <!-- afterbegin -->
      text
      <!-- beforeend -->
    </div>
      <!-- afterend -->
    </body>
    <script>
        const $foo = document.getElementById('foo');

        $foo.insertAdjacentHTML('beforebegin', '<p>beforebe
gin</p>');
        $foo.insertAdjacentHTML('afterbegin', '<p>afterbegi
```

```
n</p>');
        $foo.insertAdjacentHTML('beforeend', '<p>beforeend
</p>');
        $foo.insertAdjacentHTML('afterend', '<p>afterend</p
>');
    </script>
</html>
```

39.6.3 노드 생성과 추가

💡요소 노드 생성

- `Document.prototype.createElement(tagName)` 메서드는 요소 노드를 생성하여 반환
- `tagName` 인수에는 태그 이름을 나타내는 문자열을 전달
- `createElement` 메서드로 생성한 요소 노드는 기존 `DOM`에 추가되지 않고 홀로 존재하는 상태로, 이후에 `DOM`에 추가하는 처리가 필요
- `createElement` 메서드로 생성한 요소 노드는 아무런 자식 노드가 없는 상태

💡텍스트 노드 생성

- `Document.prototype.createTextNode(text)` 메서드는 텍스트 노드를 생성하여 반환
- `createTextNode` 메서드로 생성한 텍스트 노드는 요소 노드의 자식 노드에 추가되지 않고 홀로 존재하는 상태로, 이후에 요소 노드에 추가하는 처리가 필요

💡텍스트 노드를 요소 노드의 자식 노드로 추가

- `Node.prototype.appendChild(childNode)` 메서드는 인수로 전달한 노드를 메서드를 호출한 노드의 마지막 자식 노드로 추가
- `childNode` 인수에는 추가할 자식 노드를 전달
- 요소 노드에 자식 노드가 하나도 없는 경우에는 `textContent` 프로퍼티를 사용하여 텍스트 노드를 추가하는 것이 더욱 간편
- 요소 노드에 자식 노드가 있는 경우 `textContent` 프로퍼티에 문자열을 할당하면 요소 노드의 모든 자식 노드가 제거되고 할당한 문자열이 텍스트로 추가되므로 주의

💡요소 노드를 DOM에 추가

- `Node.prototype.appendChild` 메서드는 텍스트 노드와 부자 관계로 연결한 요소 노드를 #fruits 요소 노드의 마지막 자식 요소로 추가

- 해당 과정에서 새롭게 생성한 요소 노드는 DOM에 추가

39.7 어트리 뷰트

- `HTML` 문서가 파싱 될 때 어티리뷰트는 어티리뷰트 노드로 변환 되어 요소 노드와 연결
- 하나의 어트리뷰트당 하나의 어트리뷰트 노드 생성
- 이때 모든 어트리뷰트 노드의 참조는 유사 배열 객체이자 이터러블인 `NamedNodeMap` 객체에 담겨서 요소 노드의 **attributes 프로퍼티에 저장**

39.7.2 HTML 어트리뷰트 조작

- `Element.prototype.getAttirbute(attributeName)` : 어트리뷰트 값을 참조
- `Element.prototype.setAttirbute(attributeName, attributeValue)` : 어트리뷰트 값을 변경
- `Element.prototype.hasAttirbute(attributeName)` : 어트리뷰트 존재하는지 확인
- `Element.prototype.removeAttirbute(attributeName)` : 특정 어트리뷰트 삭제

39.7.3 HTML 어트리뷰트 vs DOM 프로퍼티

- 요소 노드 객체에는 HTML 어트리뷰트에 대응 하는 DOM 프로퍼티가 존재
- 즉, `HTML` 어트리 뷰트는 중복 관리 되는 것처럼 보인다.
  - 요소 노드의 `attributes` 프로퍼티에서 관리하는 어트리뷰트 노드
  - 요소 노드의 `DOM` 프로퍼티

💡차이점 구분

- `HTML` 어트리뷰트의 역할을 `HTML` 요소의 초기상태를 지정, 초기 상태를 의미하며 변하지 않음
- 요소 노드는 **상태**(`state`)를 가지고 있다.
- 요소 노드는 2개이상의 상태(초기 상태 와 최신 상태)를 관리해야 한다.
- 요소 노드의 초기 상태는 어트리뷰트 노드가 관리
- 요소 노드의 최신 상태는 `DOM` 프로퍼티가 관리

💡어트리뷰트 노드

- HTML 어트리뷰트로 지정한 초기상태는 어트리뷰트 노드에서 관리
- 초기 상태 값을 취득하거나 변경 하려면 `getAttribute` / `setAttribute` 메서드를 사용

💡DOM 프로퍼티

- 사용자가 입력한 최신 상태를 관리
- 단 사용자 입력에 의한 상태 변화와 관계있는 DOM 프로퍼티만 최신 상태 값을 관리(ex- input)
- getAttribute 메서드로 취득한 값은 언제나 문자열이지만, DOM 프로퍼티로 취득한 최신 상태값은 문자열이 아닐 수 있다.

39.7.4 data 어트리뷰트와 dataset 프로퍼티

- `data` 어티리뷰트와 `dataset` 프로퍼티를 사용해 어트리뷰트와 자바스크립트 간에 데이터를 교환 가능
- `data` 어트리뷰트는 `data-` 접두사 뒤에 임의의 이름을 붙여 사용
- `data` 어트리뷰트 값은 `HTMLElement.dataset` 프로퍼티로 취득
- 존재 하지 않는 이름을 키로 `dataset` 프로퍼티에 할당하면 `HTML` 요소에 `data` 어트리뷰트가 추가

39.8 스타일

39.9.1 인라인 스타일 조작

- `HTMLElement.prototype.style` 프로퍼티
- `getter` / `setter` 가 모두 존재하는 접근자 프로퍼티로 요소 노드의 인라인 스타일을 취득하거나 추가 또는 변경

```
<!DOCTYPE html>
<html>
    <body>
        <div style="color: red">Hello World</div>
        <script>
        const $div = document.querySelector('div');
```

```
        // 인라인 스타일 취득
        console.log($div.style); // CSSStyleDeclaration {
0: "color", ... }

        // 인라인 스타일 변경
        $div.style.color = 'blue';

        // 인라인 스타일 추가
        $div.style.width = '100px';
        $div.style.height = '100px';
        $div.style.backgroundColor = 'yellow';
      </script>
    </body>
</html>
```

39.8.2 클래스 조작

- class 어트리뷰트에 대응하는 DOM 프로퍼티는 class 가 아닌 `className` 과 `classList`

💡className

- `Element.prototype.className`
- `class` 어트리뷰트의 값을 문자열로 반환
- 클래스 값이 공백으로 구분되어 있을 경우 그대로 문자열로 반환

💡classList

- `Element.prototype.classList`
- `class` 어트리뷰트의 정보를 담은 `DOMTokenList` 객체를 반환
- `DOMTokenList` 객체는 class 어트리뷰트의 정보를 나타내는 컬렉션 객체로 유사 배열이면서 이터블
- `DOMTokenList` 주요 메서드
  - `add(... className)` : 인수로 전달된 1개의 이상의 문자열을 class 어트리뷰트 값으로 추가
  - `remove(... className)` : 인수로 전달한 1개 이상의 문자열과 일치하는 class 어트리뷰트에서 삭제 (일치 하는 클래스가 없으면 무시됨)

- `item(index)` : 인수로 전달한 index에 해당하는 클래스를 class 어트리뷰트에서 반환
- `contains(className)` : 인수로 전달한 문자열과 일치하는 클래스가 class 어트리뷰트에 포함되어 있는지 확인
- `replace(oldClassName, newClassName)` : 첫 번째 인수 클래스를 두번째 클래스로 변경
- `toggle(className[, force])` : 인수로 전달한 문자열과 일치하는 클래스가 존재하면 제거하고 존재하지 않으면 추가